Reactive Probabilistic Programming

Pedro Zuidberg Dos Martires
KU Leuven
pedro.zuidbergdosmartires@cs.kuleuven.be

Sebastijan Dumančić
KU Leuven
sebastijan.dumancic@cs.kuleuven.be

Abstract

Reactive programming allows a user to model the flow of a program for event-driven streams of data without explicitly stating its control flow. However, many domains are inherently uncertain and observations (the data) might ooze with noise. Probabilistic programming has emerged as the go-to technique to handle such uncertain domains. Therefore, we introduce the concept of reactive probabilistic programming, which allows to unify the methods of reactive programming and probabilistic programming. Herewith, we broaden the scope of probabilistic programming to event-driven data.

Keywords probabilistic programming, reactive programming, declarative programming, event-driven data

1 Introduction & Motivation

The declarative programming paradigm allows the user to separate the problem definition from the inference algorithm. This is in contrast to the imperative paradigm, which asks the user to specify the exact steps towards the solution. Due to the ease of modeling problems using declarative languages, the declarative programming paradigm has been used to construct probabilistic programming languages (PPL). Representatives are, for instance, the functional PPL Anglican [9] and the logic PPL Problog [2].

A challenging task for declarative programming languages is interaction with data, both reading in and writing out, especially in the context of event-driven data streams in which a program has to be able to interact with data at any point in time. To alleviate this problem, we introduce a framework of reactive probabilistic (logic) programming (RPP). Furthermore, we show how the proposed framework can be realized in the existing probabilistic logic programming language of Dynamic Distribution Clauses (DDC) [7] and its Python wrapper *PyDC* [1].

The proposed framework has the potential to broaden the scope of PPL to event-driven applications, in which new data arrives asynchronously. This is in contrast to standard (time) dynamic models, which perform inference continuously at every time step. An example application would be website ad placement, in which a visitor acts as a new event and a decision is to be made on whether to place an ad. Another example includes train scheduling problems in which a new event might be the delay, and the actor has to reason about the possibility of conflicting train routes.

2 Reactive Probabilistic Programming

Reactive programming [1], which falls itself under the declarative programming paradigm, has recently gained traction as a paradigm for event-driven applications. A common trait of event driven-applications is that external events at discrete points in time, such as a mouse click from a user on a web page, drive the execution of a program. The increased interest has resulted in various implementations, for instance, in Haskell [3] and Scala [5]. Recently also the *ReactiveX*[2] library emerged, with bindings to various different languages. The two distinguishing features of reactive programming are:

- *Behaviors* change continuously over time and are composable first-class citizens in the reactive programming paradigm. [4]
- *Events* refer to streams of value updates to time-dependent variables (behaviors). Events occur at discrete points in time and are composable first-class citizens. [1]

Mapping the concepts of behaviors and events to probabilistic programming yields the first two components of **reactive probabilistic** programming:

1. behaviors are *random variables* whose value assignments change with a transition model
2. events are *observations* which interact with the random variables through (probabilistic) observations

Reactive programming is typically used to continuously update the properties of a data structure, e.g. a website, given certain events (observations). Porting this to the probabilistic setting, we identify a third component of **reactive probabilistic** programming:

3. a planer that decides which action to take given a probabilistic world state

Given the probabilistic nature of the behaviors, events and the planner on the one hand, and the deterministic effects of actions on the other, we propose to structure reactive probabilistic programming frameworks into two modules:

- A **declarative** module, where random variables (behaviors), observations (events) and a planer are declared.
- An **imperative** module, where the effect of actions are defined.

[1] https://github.com/ML-KULeuven/PyDC

[2] http://reactivex.io/

PROBPROG 2018, October 04–06, 2018, Boston, MA, USA
2017.

```
%facts
city(brussels) <- true. %true implies the city of Brussels exists in our databse
%initial state
weather(C):0 ~ finite([0.6:rainy,0.4:sunny]) <- city(C). %we initialize the weather at time step 0
%state model
temperature(C):t ~ gaussian(10,6) <- weather(C):t ~= rainy. %given rainy weather the mean of the
                                                            %temperature is 10 degrees Celsius
temperature(C):t ~ gaussian(24,8) <- weather(C):t ~= sunny. %and 24 for sunny weather
%transition model: here we describe how variables at time (t+1) depend on variables at time (t)
weather(C):t+1 ~ finite([0.7:rainy,0.3:sunny]) <- weather(C):t ~= rainy.
weather(C):t+1 ~ finite([0.4:rainy,0.6:sunny]) <- weather(C):t ~= sunny.
activity(tintin):t+1 ~ finite([0.1:walk,0.4:shop,0.5:clean]) <- weather(brussels):t+1 ~= rainy.
activity(tintin):t+1 ~ finite([0.6:walk,0.3:shop,0.1:clean]) <- weather(brussels):t+1 ~= sunny.
```

DDC Block 1. A simple hidden Markov model describing the weather in Brussels.

3 Dynamic Distributional Clauses

Using an established probabilistic logic programming language has the advantage of inheriting an existing well defined semantics. To this end, we embody the proposed framework into the language of Dynamic Distributional Clauses [3], an extension of the logic programming language Prolog [8]. DDC is a template language that defines conditional probabilities for discrete and continuous random variables, which carry a time label.

Consider the example program in DDC Block 1, modeling the activity undertaken by Tintin based on the temperature in Brussels using a simple hidden Markov model. Having observed Tintin doing shopping (the event), we can query the weather being sunny (the behavior).

DDC carries out inference by deploying importance sampling combined with backward reasoning, likelihood weighting and Rao-Blackwellization [7]. The filtering is carried out through particle filtering [6].

4 RPP with DDC and PyDC

We showcase now the structure of an RPP framework, consisting of a declarative and an imperative module. We initialize a probabilistic belief of the world (in line 3 in Python Block 1) by loading our world model described in DDC Block 1. We then pass on the information that we observe Tintin cleaning the house and query the probabilistic module of our framework for the probability of it being hot. Based on this we take a decision on whether to wear pants or shorts.

5 Conclusions

We introduced the concept of reactive probabilistic programming and showcased an implementation utilizing DDC and the PyDC tool. In future work we would like to investigate real-world applications of this new paradigm and include additional features such as a fully fledged probabilistic planner.

```
from pydc import DDC
#load DDC program and intialize 500 particles
ddc = DDC("weather_brussels_hmm.pl", 500)
#proceed one time step and query the state
ddc.step(observations=
   "observation(activity(tintin))~=clean")
p_hot = ddc.query(
   "current(temperature(brussels))>20")
#take decision
if p_hot>0.5: print("wear shorts!")
else: print("wear pants!")
```

Python Block 1. Example code present in the imperative module of a reactive probabilistic programming framework.

Acknowledgments

Pedro Zuidberg Dos Martires is supported by the ReGround project [4], a CHIST-ERA, EU H2020 framework program and the Research Foundation - Flanders. Sebastijan Sebastijan Dumančić is supported by Research Fund KU Leuven (GOA/13/010) and the Research Foundation - Flanders (G079416N).

References

[1] Engineer Bainomugisha, Andoni Lombide Carreton, Tom van Cutsem, Stijn Mostinckx, and Wolfgang de Meuter. 2013. A Survey on Reactive Programming. *ACM Comput. Surv.* 45, 4, Article 52 (Aug. 2013), 34 pages. https://doi.org/10.1145/2501654.2501666

[2] Anton Dries, Angelika Kimmig, Wannes Meert, Joris Renkens, Guy Van den Broeck, Jonas Vlasselaer, and Luc De Raedt. 2015. ProbLog2:

[3] https://bitbucket.org/problog/dc_problog/

[4] http://reground.cs.kuleuven.be

Probabilistic logic programming. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 312–315.

[3] Conal Elliott. 2009. Push-pull functional reactive programming. In *Haskell Symposium*. http://conal.net/papers/push-pull-frp

[4] Conal Elliott and Paul Hudak. 1997. Functional reactive animation. In *ACM SIGPLAN Notices*, Vol. 32. ACM, 263–273.

[5] Ingo Maier and Martin Odersky. 2012. *Deprecating the Observer Pattern with Scala. React.* Technical Report.

[6] Davide Nitti, Tinne De Laet, and Luc De Raedt. 2013. A particle filter for hybrid relational domains. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2764–2771.

[7] Davide Nitti, Tinne De Laet, and Luc De Raedt. 2016. Probabilistic Logic Programming for Hybrid Relational Domains. *Mach. Learn.* 103, 3 (June 2016), 407–449. https://doi.org/10.1007/s10994-016-5558-8

[8] Leon Sterling and Ehud Y Shapiro. 1994. *The art of Prolog: advanced programming techniques*. MIT press.

[9] Frank Wood, Jan Willem van de Meent, and Vikash Mansinghka. 2014. A New Approach to Probabilistic Programming Inference. In *Proceedings of the 17th International conference on Artificial Intelligence and Statistics*. 1024–1032.